Pergamon

Computers Math. Applic. Vol. 34, No. 9, pp. 39–49, 1997

0898-1221/97 $17.00 + 0.00

PII: S0898-1221(97)00187-9

Fast Parallel Algorithms for Forecasting

P. K. JANA
Department of Computer Science and Engineering, Birla Institute of Technology
Mesra, Ranchi 835 215, India

B. P. SINHA
Electronics Unit, Indian Statistical Institute
Calcutta 700 035, India

(Received October 1995; accepted April 1997)

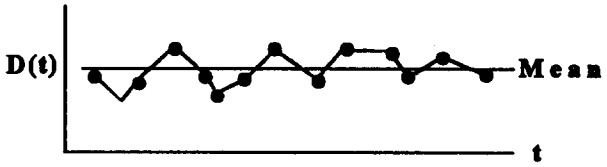
Abstract—This paper presents two parallel algorithms for forecasting implemented on a linear array and a tree model [1]. Both the algorithms are based on the weighted moving average technique [2,3]. Given that m and n are the numbers of the input observed data values and the numbers of weights, respectively, the algorithm on a linear array of n processors requires $m + 1$ steps and that on a tree model with $(2n - 1)$ processors (n being a power of 2), needs $(m - n + 2) + \log_2 n$ steps. It has also been shown how the corresponding algorithms can be extended to the case when the number of available processors is less than n (for a linear array) or $2n - 1$ (for a tree model). The corresponding algorithms mapped on an ST-array (Store and Trigger array with p processors, $p \leq n$) [4] and an ST-tree (Store and Trigger tree with $2p - 1$ processors, $p \leq n$, p being a power of 2) require $n/p(m - n + 1) + p - 1$ and $n/p[(m - n + 2) + \log_2 p]$ steps, respectively.

Keywords—Forecasting, Weighted moving average, Matrix by vector multiplication, Linear array, Tree model.

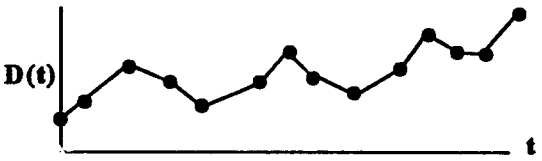
1. INTRODUCTION

Forecasting usually means predicting a future event from the information on a set of past events. There are many kinds of forecasting, e.g., forecasting of economy, prices, interest rates, business conditions, technology, weather, etc. In the commercial field, demand forecasting plays an important role and it is an integral part of management information systems, especially the inventory control systems. Among the different quantitative forecasting models [3] available for successful implementation of decision making systems, the time series models are very popular. In these models, if we have an observed time series say, $d(1), d(2), \ldots, d(m)$, where $d(i)$ represents the data at the i^{th} time instant $1 \leq i \leq m$, then the problem of forecasting is to estimate $d(m+1)$ or, more generally, $d(m+\tau)$, where τ is a small positive integer. All forecasting methods assume that some pattern or relationship exists that can be identified and that pattern is used as the basis for forecasting. For the qualitative methods of forecasting [3], these patterns can take virtually any form and need not be identified explicitly. In quantitative methods of forecasting, however, each technique does make explicit assumptions about the type of underlying pattern. Therefore, the ability of a given technique to forecast effectively in a specific situation depends largely on matching the pattern in that situation with a technique that can handle that pattern [5]. In other words, different models may be applied to the same data pattern, but one model will usually fit the data in the best best possible manner. For example, a simple moving average usually provides a good fit for a constant process such as that shown in Figure 1a. If the plot of data is cyclical around a constant trend, a weighted moving average is much more suitable. Similarly,

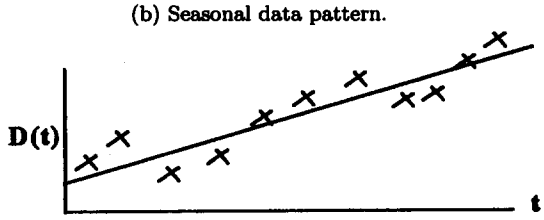
an exponential smoothing model is the best to use if the plot of data exhibits a constant process, with or without cyclical tendencies [2]. The four types of commonly observed data patterns have been shown in Figure 1.



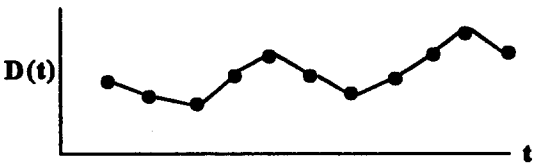
(a) Constant data pattern.



(b) Seasonal data pattern.



(c) Tread data pattern.



(d) Cyclical data pattern.

Figure 1. Common types of data patterns.

Evans and Gusev [6] have presented systolic implementations of a moving average (MA) filter with the minimal time steps of $m + n + 1$ on $n + 1$ processors.

In this paper, we propose two systolic parallel algorithms for forecasting implemented on a linear array and a tree model. Both the algorithms are based on the weighted moving average technique. Given that m and n are the numbers of the input observed data values and weights, respectively, we show that

(1) the algorithm on n processors connected in the form of a linear array requires $m+1$ steps, and
(2) the algorithm on the tree model using $2n - 1$ processors, n being a power of 2, requires $m - n + 2 + \log_2 n$ steps for execution.

We then show how the corresponding algorithms can be extended to the cases when the number of available processors is less than n (for the linear array model) or $2n - 1$ (for the tree model). These algorithms mapped on ST-array (Store and Trigger array) with p processors ($p \leq n$) [3] and ST-tree (Store and Trigger tree) with $2p - 1$ processors, where $p \leq n$ and p is a power of 2, require $n/p(m - n + 1) + p - 1$ and $n/p[(m - n + 2) + \log_2 p]$ steps, respectively.

The paper is organized as follows. A sequential algorithm for the weighted moving average is discussed in Section 2, followed by the two parallel algorithms in Section 3.

2. WEIGHTED MOVING AVERAGE

For short term forecasting, the weighted moving average is a widely accepted technique. If the plot of data exhibits a cyclical pattern around a constant trend, the weighted moving average is the best to use as a forecasting method. The method is briefly described as follows.

Given a set of positive weights say, $w(1), w(2), \ldots, w(n)$ for the n most recent observations $d(t), d(t-1), \ldots, d(t-n+1)$, the weighted moving average of these observations at time t is given by

$$W^M(t) = \frac{w(n)\,d(t) + w(n-1)\,d(t-1) + \cdots + w(1)\,d(t-n+1)}{w(n) + w(n-1) + \cdots + w(1)},$$

where $0 \le w(1) \le w(2) \le \cdots \le w(n)$. The weighted average $W^M(t)$ is used as the forecast value $\hat{d}(t+\tau)$ at the time $t+\tau$, that is, $\hat{d}(t+\tau) = W^M(t)$, τ being a small positive integer. Given m past observed data values $d(1), d(2), \ldots, d(m)$ and n $(n \le m)$ fixed weights $w(1), w(2), \ldots, w(n)$, $m-n+1$ weighted moving averages can be computed by shifting the window of size n over the data set $d(1), d(2), \ldots, d(m)$. The corresponding mean square error of forecasting can then be defined as

$$\text{MSE} = \sum_{t=n+\tau}^{m} \frac{\left[d(t) - \hat{d}(t)\right]^2}{(m-n-\tau+1)}.$$

As an example, let $\tau = 1$. Then, for a fixed value of n, the $m-n+1$ weighted moving averages and the corresponding errors in prediction are computed as follows.

Weighted Moving Average	Error
$W^M(n) = \dfrac{w(1)\,d(1) + w(2)\,d(2) + \cdots + w(n)\,d(n)}{w(1) + w(2) + \cdots + w(n)}$	$E(n+1) = d(n+1) - W^M(n)$
$W^M(n+1) = \dfrac{w(1)\,d(2) + w(2)\,d(3) + \cdots + w(n)\,d(n+1)}{w(1) + w(2) + \cdots + w(n)}$	$E(n+2) = d(n+2) - W^M(n+1)$
$W^M(n+2) = \dfrac{w(1)\,d(3) + w(2)\,d(4) + \cdots + w(n)\,d(n+2)}{w(1) + w(2) + \cdots + w(n)}$	$E(n+3) = d(n+2) - W^M(n+2)$
$\vdots$	$\vdots$
$W^M(m) = \dfrac{w(1)\,d(m-n+1) + w(2)\,d(m-n+2) + \cdots + w(n)\,d(m)}{w(1) + w(2) + \cdots + w(n)}$	$E(m) = d(m) - W^M(m-1)$

$$\text{MSE} = \sum_{i=n+1}^{m} \frac{E_i^2}{m-n}.$$

The objective is to choose the optimum value of n so that the MSE is minimized. In practice, using different values of n, say $n_1, n_2, \ldots, n_q$, along with a set of predefined weights, the weighted moving averages are computed. The value of n which gives the least MSE is chosen for forecasting the value $\hat{d}(t+\tau)$.

2.1. Special Cases

CASE A. SIMPLE MOVING AVERAGE. This method is most appropriate for a constant process as shown in Figure 1a. In this case, equal weights are given to all observations. Thus, each weight is set to $1/n$ and the corresponding weighted average is given by

$$S^M(t) = \frac{d(t) + d(t-1) + \cdots + d(t-n+1)}{n}.$$

CASE B. SIMPLE EXPONENTIAL SMOOTHING. A major shortcoming of the simple moving average technique is that it assigns equal weights to all the past observations, leading to a larger

error in prediction. On the other hand, in the simple exponential smoothing technique, the more recent observations are given larger weights than the older ones and this leads to a smaller error in prediction than the simple moving average technique. Because of this, exponential smoothing has gained wider acceptance as a forecasting method. This technique is recommended when the data exhibits a constant process, with or without cyclical tendencies. Mathematically, we express it in terms of weighted average as

$$E^M(t) = \alpha\, d(t) + \alpha(1-\alpha)\, d(t-1) + \alpha(1-\alpha)^2\, d(t-2) + \cdots + \alpha(1-\alpha)^{n-1}\, d(t-n+1),$$

where the weights $w(i) = \alpha(1-\alpha)^{n-i}$, $1 \le i \le n$, and $0 \le \alpha \le 1$. Clearly, the weights $w(1)$, $w(2), \ldots, w(n)$ are in decreasing order of magnitude. The value of α determines the relative values of these weights. A large value of α (close to 1.0) assigns most of the weight to the recent observations and vice versa. When a value of α close to 1.0 gives the best result, it may indicate that trends or seasonalities are present and that simple exponential smoothing is not the best approach [5]. Exponential smoothing has been applied extensively in a number of business situations. It is easy to understand, straightforward to apply, and intuitively appealing to a manager, because he has some control over the weights through assigning a value to α. A major drawback of this method, however, is that there is no easy way to determine an appropriate value of α.

The sequential algorithm for computing the weighted moving averages with $\tau = 1$ and a given value of n runs as follows.

2.2. Sequential Algorithm

Input : $d(1)$, $d(2)$, $\cdots$, $d(m)$; $w(1)$, $w(2)$, $\cdots$, $w(n)$
output: $W^M(n)$, $W^M(n+1)$, $\cdots$, $W^M(m)$, MSE

1. weight_sum := 0; *sum* := 0; *square_error* := 0
2. **for** $i = 1$ to n **do**
   weight_sum := *weight_sum* + $w(i)$
   endfor
3. **for** $i = n$ **to** m **do**
   begin
   sum := 0;
   for $j = i - n + 1$ **to** i **do**
   sum := *sum* + $w(n + j - i) * d(j)$
   $W^M(i)$:= *sum*/*weight_sum*
   endfor
   if $i < m$ **then** *square_error* := *square_error* + $(d(i+1) - W^M(i))^2$
   end
4. MSE = *square_error*/$(m - n)$

3. PARALLEL ALGORITHMS

From the previous section, it is clear that for different values of n say, n_1, $n_2, \ldots, n_q$, the weighted moving averages are to be computed and it requires q iterations. For $n = n_i$, $1 \le i \le q$, we need to compute $m - n_i + 1$ weighted moving averages. In the parallel implementation of the weighted moving average technique, we exploit the parallelism among different steps to reduce the execution time of the weighted moving averages per iteration. In this section, we present two parallel algorithms implemented on two systolic architectures, one on a linear array and the other on a tree. We would also show how the corresponding algorithms can be implemented on an ST-array (with p processors, $p \le n$) and an ST-tree with $2p-1$ processors, where p is a power of 2 and $p \le n$.

Let the weighted moving averages $W^M(n+i)$, $0 \leq i \leq m-n$, as given in Section 2, be represented as

$$W^M(n+i) = \frac{N^M(n+i)}{\Delta^M(n+i)},$$

where N^M and Δ^M stand for the numerator and the denominator of W^M. Hence, $N^M(n+i)$'s can be written in terms of the following matrix by vector multiplication:

$$\begin{pmatrix} N^M(n) \\ N^M(n+1) \\ N^M(n+2) \\ \vdots \\ N^M(m) \end{pmatrix} = \begin{pmatrix} d(1) & d(2) & \cdots & d(n) \\ d(2) & d(3) & \cdots & d(n+1) \\ d(3) & d(4) & \cdots & d(n+2) \\ \vdots & & & \vdots \\ d(m-n+1) & d(m-n+2) & \cdots & d(m) \end{pmatrix} \begin{pmatrix} w(1) \\ w(2) \\ w(3) \\ \vdots \\ w(n) \end{pmatrix} = D * W,$$

where D is the $(m-n+1) \times n$ data matrix and W is the $n \times 1$ weight vector as given above.

The denominator $\Delta^M(n+i)$ is equal to $w(1)+w(2)+\cdots+w(n)$ for all values of i.

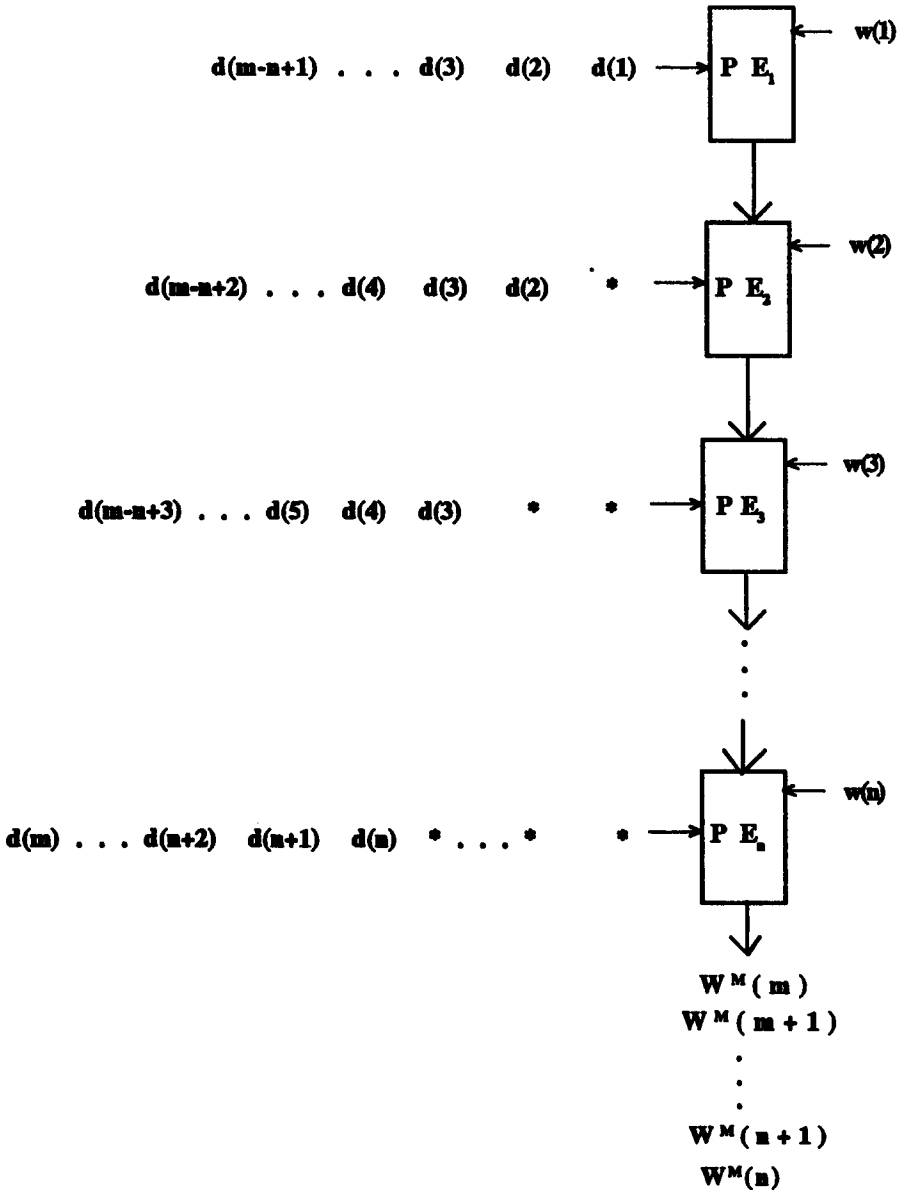


Figure 2. Computation on the linear array.

3.1. Implementation on Linear Array

We use a linear array of n processors $PE_1, PE_2, \ldots, PE_n$ as shown in Figure 2. We assume that every processor PE_i, $1 \leq i \leq n$ has five registers DR_i, WR_i, NR_i, LR_i, and VR_i. Registers DR_i and WR_i will be used for storing the input data and weights, respectively. The register VR_i will be used for communication with the processors PE_{i-1} and PE_{i+1} in the vertical direction, when they exist. Initially, LR_i and NR_i, $1 \leq i \leq n$, are set to zero. The weights $w(1), w(2), \ldots, w(n)$ are given to the WR-registers of the processors $PE_1, PE_2, \ldots, PE_n$ once for all. The j^{th} column elements of the D-matrix, $1 \leq j \leq n$ are fed successively to the DR-register of the processor PE_j with an initial delay of $(j-1)$ units for synchronization purposes as shown in Figure 2 (a star-mark (*) in Figure 2 indicates a delay of one unit time). The algorithm is described below.

ALGORITHM A.

Step 1 : **for** $j := 1$ to n **do in parallel** /* *initialization* */
begin
load $w(j)$ to WR_j register of the processor PE_j;
$VR_j := WR_j$; $LR_j := 0$; $NR_j := 0$
end.

Step 2 : **for** $i := 1$ to m **do**
begin
for $j := 1$ to n **do in parallel**
begin
for $j \leq i \leq m-n+j$, PE_j does the Steps 2.1 and 2.2 **in parallel**
2.1 it sends the content of its VR_j to VR_{j+1} of PE_{j+1};
2.2 it loads the next (appropriately delayed) input data element,
that is, DR_j receives the data $d(i)$;
$NR_j := NR_j + DR_j * WR_j$;
if $i = j-1$ **then** $LR_j := LR_j + VR_j$
else if $j \leq i \leq m-n+j$ **then** $NR_j := NR_j + VR_j$;
$VR_j := NR_j$;
end;
if $i \geq n$ **then** $NR_n := NR_n / LR_n$
and PE_n outputs the content of NR_n as $W^M(i)$;
end.

/*Eventually, the moving averages emerge from the processor PE_n */

Time complexity

Step 1 of Algorithm A takes a constant time. After this initialization phase of Step 1, the loop in Step 2 is executed m times. Let t_i, t_c, t_a, t_m, t_r, and t_d be the times required for inputting one data element, one data communication, one addition, one multiplication, one register transfer, and one division, respectively. Hence, Algorithm A can be completed in $m+1$ basic execution cycles where each cycle needs $t_i + t_c + t_r + 2t_a + t_m + t_d$ time, i.e., the time complexity of this algorithm is $O(m)$.

3.1.1. Implementation on ST-array

The previous linear array is applicable for the cases when the number of columns of the matrix D is less than or equal to the array size. But when n exceeds the number of processors, we implement the algorithm on an ST-array [3] as follows.

Let us assume that $n = pr$, $p > 1$, $r > 1$, and p is the number of available processors. Here, we use another control register CR_i for every processor PE_i, $1 \leq i \leq p$. Initially, LR_i and NR_i, $1 \leq i \leq p$, are both set to zero. The computation can be done in two phases. In the first phase,

the following elements are inputted to WR_i and CR_i registers of each PE_i, after an initial delay of $(i-1)$ units for synchronization purposes.

(1) A sequence of weights $w(i)$, $w(i+p)$, $w(i+2p),\ldots,w(i+(r-1)p)$, is successively inputted to the WR_i register.

(2) A sequence of $(r-1)$ 0's followed by a '1', is successively inputted to the CR_i register.

Each time an input is received by the WR_i and CR_i registers, PE_i, $1 \le i \le p$ performs the following operations in order.

(1) The content of LR_i is updated as
$LR_i := LR_i + WR_i$.

(2) If $CR_i = 1$ then
if $i < p$ then
the content of LR_i of PE_i is transmitted to
the register LR_{i+1} of PE_{i+1}

/* After all inputs are consumed by PE_p, the the register LR_p will contain the sum $w(1)+w(2)+\cdots+w(n)$ at the end of the first phase */

In the second phase, the following inputs are given to each PE_i.

(1) The data elements $d(i)$, $d(i+p)$, $d(i+2p),\ldots,d(i+(r-1)p$; $d(i+1)$, $d(i+p+1)$, $d(i+2p+1),\ldots,d(i+(r-1)p+1);\ldots,d(i+p-1)$, $d(i+2p-1)$, $d(i+3p-1),\ldots,d(i+rp-1)$ are successively inputted to the DR_i register.

(2) A repetitive sequence of weights $w(i)$, $w(i+p)$, $w(i+2p),\ldots,w(i+(r-1)p)$ is successively inputted to the WR_i register. This sequence is repeated so long as there will be an input to the DR_i register.

(3) A repetitive sequence of $(r-1)$ 0's followed by a '1', is successively inputted to the CR_i register. This sequence is repeated so long as there will be an input to the DR_i register.

Each time an input is received by each of the DR_i, WR_i, and CR_i registers, PE_i, $1 \le i \le p$ performs the following operations in order:

(1) The content of NR_i is updated as
$NR_i := NR_i + WR_i * DR_i$.

(2) If $CR_i = 1$ then
if $i < p$, then
the content of NR_i of PE_i is transmitted to
the register NR_{i+1} of PE_{i+1} and after that
NR_i is reset to zero.
else /* i = p */
$NR_i := NR_i/LR_i$, the value of NR_i is ouputted
and NR_i is reset to zero.

/* At the end of this step, the weighted moving averages emerge from the NR_p register of the processor PE_p */

In fact, the execution of phase 1 can easily be overlapped with that of phase 2 by using an additional flag in each processor, which can be initialized to '1' and will be reset to '0' when a '1' is received in the CR_i. Execution of both phase 1 and phase 2 will be started at the same time, but phase 1 execution will not be repeated any more when this flag is reset to '0'.

The method is illustrated with an example for $m = 11$ and $n = 9$ as shown in Figure 3 (a star-mark (*) in Figure 3 indicates a delay of one unit time).

Time complexity

The processor PE_p gets the first input data after an initial delay of $p-1$ steps. After that, PE_p computes each of the $m-n+1$ weighted moving averages $W^M(n)$, $W^M(n+1),\ldots,W^M(m)$ in $r = n/p$ steps. Hence, by overlapping the executions of phase 1 and phase 2, it turns out that the total computation can be done in $n/p(m-n+1)+p-1$ steps.

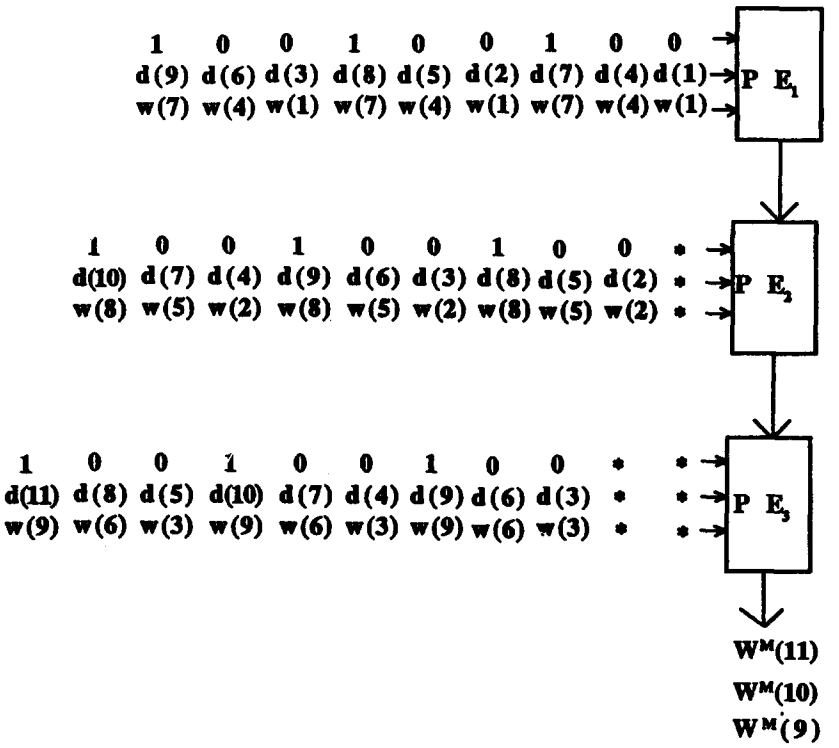


Figure 3. Computation of the ST-array.

3.2. Implementation on Tree

In order to do the same job, we can also use a complete binary tree of $2n-1$ processors (we assume that n is a power of 2). The arrangement of the processors for $m=9$ and $n=4$ is shown in Figure 4 with n processors $PE_1, PE_2, \ldots, PE_n$ at the leaf nodes.

In the first step of the algorithm, every leaf processor PE_i, $1 \leq i \leq n$ is given the input $w(i)$ to be stored in its WR_i register. These weights are summed up using the binary tree connections and the result is stored in the LR-register of the root processor. The LR-register of the root processor now contains the sum $w(1)+w(2)+\cdots+w(n)$.

In the second step of the algorithm, the i^{th} column elements of the D matrix are successively inputted to the leaf processor PE_i, $1 \leq i \leq n$, as shown in Figure 4. When the leaf processor PE_i receives an input $d(j)$, it computes the product $d(j).w(i)$ and sends the result to its parent. When an intermediate processor receives two inputs from both of its children, it adds them and sends the result to its parent. When the root processor receives two inputs from both of its children, it adds them, divides the sum by the content of its LR-register and outputs the result as the weighted moving average.

Time complexity

Since there are $m-n+1$ rows and n is the number of leaf processors, then the first step of the algorithm requires $\log_2 n$ steps and the second step requires $(m-n+1)+\log_2 n$ steps. However, it is easy to pipeline the operations in phase 1 and phase 2 so that the algorithm can be completed in $1+(m-n+1)+\log_2 n = (m-n+2)+\log_2 n$ steps.

3.2.1. Implementation on ST-tree

When we have a fewer number of leaf processors than the number of columns in the D matrix, the store and trigger method can be applied for computing the weighted moving averages on the tree model as well. The method is described below.

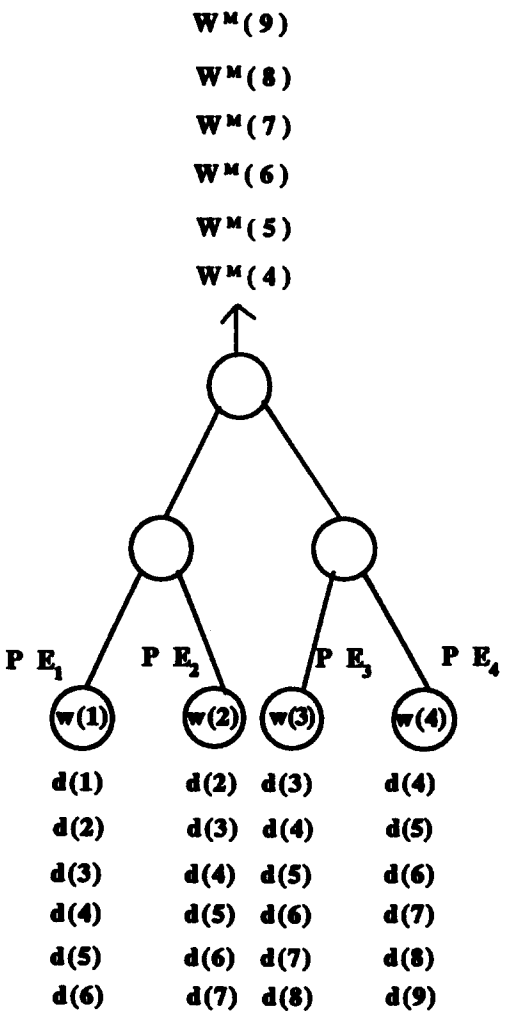


Figure 4. Computation on the tree.

Suppose $n = pr$ and we have a complete binary tree of $2p-1$ processors with p leaf processors (assume that p is a power of 2). Each of the two phases of computation described in Section 3.2 is now completed in several steps. The following operations will be done in the two phases.

1. In the k^{th} step of the first phase $1 \leq k \leq r$, the weights $w(k)$, $w(k+1), \ldots, w(k+p-1)$ are inputted in parallel to all the leaf processors. Whenever a leaf processor receives an input, it forwards that data to its parent. When an intermediate processor receives the data from two of its children, it adds them and sends the sum to its parent. If the root processor receives the data from both of its children, it adds them and stores the sum in a temporary register. After the completion of all the steps in the first phase, the root processor contains the sum $w(1) + w(2) + \cdots + w(n)$.
2. In the second phase, computation over $m-n+1$ rows of the D matrix is done in $m-n+1$ subphases to get a weighted moving average after every such subphase. However, each subphase now consists of r steps. In the k^{th} step of the j^{th} subphase, $1 \leq j \leq (m-n+1)$, $1 \leq k \leq r$, the elements D_{jk}, $D_{j,k+1}, \ldots, D_{j,k+p-1}$ of the D matrix and the weights $w(k)$, $w(k+1), \ldots, w(k+p-1)$ are inputted in parallel to all the leaf processors. Whenever the leaf processor PE_i receives these inputs, it multiplies them and sends the product to its parent. An intermediate processor adds the two inputs received from its two children, and sends the result to its parent. The root processor maintains a counter, initialized to zero and does the same job like an intermediate processor plus the following additional operations.

(i) It increments the counter by 1.

(ii) If the value of the counter is equal to r, then it divides the current sum of the products by the sum $w(1) + w(2) + \cdots + w(n)$, outputs the quotient as a weighted moving average, and resets the counter to zero; *otherwise*, it just stores the partial sum of the products in a temporary register.

The method is illustrated in Figure 5 for $m = 9$, $n = 8$, and $p = 4$, i.e., $r = 2$.

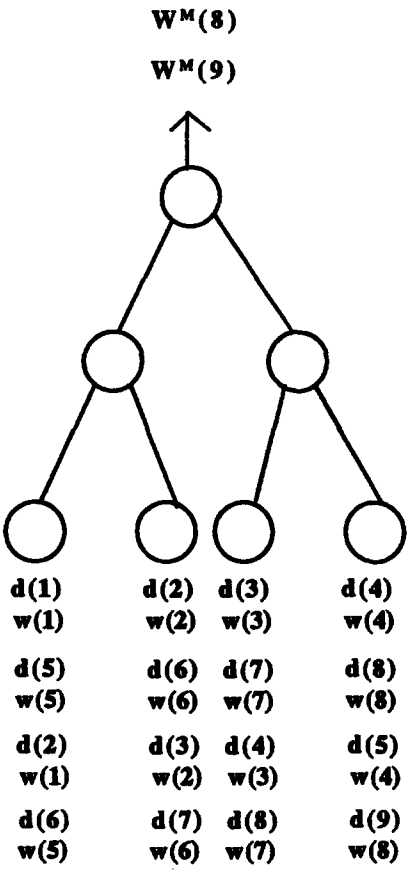


Figure 5. Computation on the ST-tree.

Time complexity

By properly pipelining the operations in phase 1, it can be completed in $n/p + \log_2 p$ steps. Similarly, through pipelining, the computations in phase 2 can be done in $n/p[(m-n+1)+\log_2 p]$ steps. Moreover, it is also possible to pipeline phase 1 and phase 2 as in the case of a tree, when the overall computation can be done in $n/p[(m - n + 2) + \log_2 p]$ steps.

4. CONCLUSION

Two parallel algorithms for short-term forecasting have been developed. The algorithms are based on the weighted moving average technique and have been implemented on a linear array and a tree. On a linear array of n processors, it requires $m + 1$ steps and on a complete binary tree of $(2n - 1)$ processors, n being a power of 2, it requires $(m - n + 2) + \log_2 n$ steps, where m is the number of input observed data values and n weights are used to compute the weighted moving averages. We have also shown how the corresponding algorithms can be extended to the cases when only fewer processors are available. These algorithms mapped on an ST-array (Store and Trigger array with p processors) and an ST-tree (Store and Trigger tree with $2p - 1$ processors, p being a power of 2) require $n/p(m - n + 1) + p - 1$ and $n/p[(m - n + 2) + \log_2 p]$ steps, respectively.

REFERENCES

1. S.G. Akl, *The Design and Anlysis of Parallel Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, (1989).
2. J. Banks, J.P. Spoerer and R. Collins, *IBM PC Applications for the Industrial Engineer and Manager*, Prentice-Hall, Englewood Cliffs, NJ, (1986).
3. S.C. Wheelwright and S. Makridakis, *Forecasting Methods for Management*, Third edition, John Wiley and Sons, New York, (1980).
4. Y.-C. Lin, Array size anomaly of problem-size independent systolic array for matrix-vector multiplication, *Parallel Computing* **17** (4/5), 515–522 (July 1991).
5. S. Makridakis and S.C. Wheelwright, *Interactive Forecasting*, Second edition, Holden-Day, San Francisco, CA, (1978).
6. D.J. Evans and M. Gusev, New linear systolic arrays for digital filters and convolution, *Parallel Computing* **20** (1), 29–61 (1994).